Ruddy Castro

Ivy Nguyen

David Kwon

P2P Project

Overall, the project was extremely challenging. As of now, the project is still incomplete, but major the logic for the program is present. There are several issues listed below that we encountered, some which we simply ran out of time to get to since we kept getting bugs we needed to fix.

The biggest issue we ran into, and are still having trouble with, was being able to connect from a client that is not the same machine. There were two issues that would occur when trying to connect. The first was that the client would not see the server and then would take the position of the server. The second was I would get an error stating that the connection was refused, which we believed could have been a firewall issue.

Another issue we ran into was getting the threads for each machine to work properly. For most of the time spent, our client and server would run the code in their constructor to completion, given no bugs occurred, and then continue with the P2P main loop. This caused the machine to take on the other role, which we needed a check in place to stop this from happening. After a lot of debugging, we realized that the main thread was continuing after creating the supporting threads for the given role. This was causing errors to occur since the machine was swapping roles in the middle of trying to carry out tasks given. We fixed this error, partially, by adding the join() method for supporting threads to stop the main thread from carrying out until the child thread finished their respective task. Further testing is still needed to see if any bugs come up after implementation of other features.

We were unable to finish the part of the program that saves the files to the directory. Initially, we hard coded all the information that we wanted to test the relationship with for client and server. We were able to successfully send over data from one to the other using the created methods. Once we expanded the code to include distributed hash tables (DHT) and removed the hard coded paths, we once again ran into issues saving the data. We are currently dealing with the issue of saving the file to the directory since we have not gone in to update the supporting methods.

The way we envisioned the program to run is that the first machine to connect, automatically becomes the server and awaits a connection from a client. The server for us will be the workhorse of the program that handles all the file comparison. When a new machine connects, it will assume the role of client and start its run method after successfully connecting with out server. In Code #1, we can see the code that the client function will run once created.

```
def run(self):
        self.initialConnection()
        try:
            i_thread = threading.Thread(target=self.waitForCompare)
            i_thread.start()
            i_thread.join()

            time.sleep(1)

        except:
            print("Failed compare")

    def waitForCompare(self):
        while True:
            time.sleep(2)
            code = self.s.recv(1024).decode('utf-8')

            if code == "s":
                # need to send file over
                fileName = self.s.recv(1024).decode('utf-8')
                msg = self.dht[fileName][0]
                self.s.send(msg)

            elif code == "r":
                fileName = self.s.recv(1024).decode('utf-8')
                # will receive file
                fileContent = self.s.recv(1024)
                self.receive_message()

            elif code == "q":
                # no more data, we can leave
                break

        self.dht()


    def initialConnection(self):
        # Send over DHT
        dict_DHT = pickle.dumps(self.fileList())

        self.s.sendall(dict_DHT)
```

Code #1: Client run, waitForCompare, initialConnection functions.

What we notice here is that the Client is sending over its DHT to the server using the pickle class. After it sends the DHT, the server will use it to start the comparison function listed below in Code #2.

```
"""
    Code that compares the client dht to that of the server based off three thing
s by the following priority
    1.filename = key
    2.hash value (if the keys are the same but different hash values)
    3.most recent timestamp (the more recent file will be the one sent to the oth
er machine)]
    """
    def comparedht(self, clientdht):

        # initiate a temp dictionary to hold file
        #tempDict = {}

         # start comparing files
        for filename in clientdht.keys():
            # same name:
            # keys() returns a list of all the available keys in the clientdht
            if filename in self.dht.keys():
                # pass if the files have the same hash value
                if self.dht[filename][0] == clientdht[filename][0]:
                    # No changes were made
                    # move on to the next file
                    continue

                else:
                    # Need to compare time stamps
                    if self.dht[filename][1] > clientdht[filename][1]:
                        # Server has most up to date
                        # Send file to client
                        self.connections[0].send("r".encode('utf-8'))
                        time.sleep(1)
                        self.connections[0].send(filename.encode('utf-8'))
                        msg = self.dht[filename][0]
                        self.connections[0].send(msg)

                    else:
                        # Client has most up to date
                        # Request file from client
                        self.connections.send("s".encode('utf-8'))
                        time.sleep(1)
                        self.connections[0].send(filename.encode('utf-8'))

                        fileContent = self.connections[0].recv(BYTE_SIZE)

                        P2P.makefile(fileContent)
```

```
            else:
                # File not in server directory
                # Client sends file over
                self.connections[0].send("s".encode('utf-8'))
                time.sleep(1)
                self.connections[0].send(filename.encode('utf-8'))

                fileContent = self.connections[0].recv(BYTE_SIZE)

                # write data to directory
                P2P.makefile(fileContent)

        # adding leftover file in client node to the temp dictionary
        for filename in self.dht:
            if filename not in clientdht.keys():
                # Client needs file from server
                # server sends file to client
                self.connections[0].send("r".encode('utf-8'))
                time.sleep(1)
                self.connections[0].send(filename.encode('utf-8'))
                msg = self.dht[filename][0]
                self.connections[0].send(msg)



        self.s.send('q'.encode('utf-8'))
        self.fileList() # updating server dht
```

Code #2: Server's comparedht function.

The server will take the clientdht and start comparing the values with that of its own DHT. The DHT has a format of the file full name being the key and the content inside being the hash for it. An extra field was added to the DHT to include the last altered time in the form of a float using os.path.getmtime(). We started the function with a for loop that grabbed the first filename from the client's DHT. From there we checked to see if that file was present in the server's dht using an if statement. If the file was present, we had to check to see if the content was changed. The only significant code came if the hash values were different, indicating that the content has been altered. If the content was altered, we checked the timestamp by using the float value added. The greater float dictated the file that was most recently altered. From there we either sent or received data from the client.

The client knew whether or not to send data by checking the initial code that the server sent. The code was either an "s", "r", or "q" which stood for send, receive, or quit. From here, the client used that code in waitForCompare in Code #1. The data was then either received or sent to the server, leading to saving the file. This is where we unfortunately ran out of time and were unable to finish up the rest of the project.